Formal Verification of SIGNAL Programs: Application to a Power Transformer Station Controller

Michel Le Borgne, Hervé Marchand, Eric Rutten, Mazen Samaan

Formal Verification of SIGNAL Programs: Application to a Power Transformer Station Controller*

Michel Le Borgne, Hervé Marchand, Éric Rutten
IRISA / INRIA - Rennes
F-35042 RENNES, France
e-mail:{leborgne,hmarchan,rutten}@irisa.fr

Mazen Samaan
EDF/DER, EP, dept. CCC
6 quai Watier, 78401 CHATOU, France
e-mail: Mazen.Samaan@der.edf.fr

Abstract. We present a methodology for the verification of reactive systems, and its application to a case study. Systems are specified using the synchronous data flow language SIGNAL. As this language is based on an equational approach (*i.e.* SIGNAL programs are constraint equations between signals), it is natural to translate its Boolean part into a system of polynomial equations over three values denoting *true*, *false* and *absent*. Using operations in algebraic geometry on the polynomials, it is possible to check properties concerning the system, such as liveness, invariance, reachability and attractivity. We apply this method to the verification of the automatic circuit breaking control system of an electric power transformer station. This system handles the reaction to electrical defects on high voltage lines.

Keywords: Reactive systems, synchronous language, verification, case study.

1 Introduction

This paper presents a formal method for the verification of reactive real-time systems and its application to the case study of the controller of a power transformer station. The specification of the controller is made in the real-time synchronized data-flow language SIGNAL[10]. Its declarative style is based on equations defining the values and the synchronizations of flows of data called signals. Schematically, processes are systems of equations, and the compilation of a SIGNAL program involves transforming the specification into an executable code solving this system of equations at each reaction. Compilation performs the checking of the causal and temporal consistency of the specification. Some statical properties can thus be proved by the compiler (this part of the verification is only briefly mentioned in this paper; see [2] and [10] for details). The original equational nature of SIGNAL makes that it relies on a formal model in terms of polynomial dynamical equations systems, and the proof method is based on the theory of algebraic geometry. This way, it is possible to prove a wide variety of dynamical properties, such as *liveness*, *invariance*, *reachability* or *attractivity* [9, 5]. This paper focuses on the method for verification, based on this model, and on its application to a case study.

The formal method is applied to the verification of the automatic circuit-breaking controller of an electric power transformer station. It concerns the response to electric

* This work is supported by Électricité de France (EDF).

defects on the lines traversing it. The functionality of the controller is to handle the interruption of current, the redirection of supply sources, and the re-establishment of current following an interruption. The objective is double: protecting the components of the transformer itself, and minimizing the defect in the distribution of power in terms of duration and size of the interrupted sub-network. This system has been specified in SIGNAL and SIGNAL*GTi*, which is a extension of the language, with the notion of time intervals and preemptive tasks [11].

The remainder of this paper is organized as follows: Section 2 presents an outline of the data-flow language SIGNAL, and of its model in polynomial dynamical systems. The algebraic method dedicated to the verification of SIGNAL programs is described in Section 3. Their application to the verification of the controller of a transformer station is described in Section 4. Discussion on results and related work is given in Section 5.

2 The SIGNAL language and its model

2.1 The SIGNAL equational language

SIGNAL [10] is built around a minimal kernel of operators. It manipulates *signals* `X`, which denote unbounded series of typed values $(\mathtt{x}_t)_{t\in T}$, indexed by time t in a time domain T. An associated clock determines the set of instants at which values are present. A particular type of signals called `event` is characterized only by its presence, and always has the value *true* (hence, its negation by `not` is always *false*). The clock of a signal `X` is obtained by applying the operator `event X`. The constructs of the language can be used in an equational style to specify the relations between signals i.e., between their values and between their clocks. Systems of equations on signals are built using a composition construct, thus defining *processes*. Data flow applications are activities executed over a set of instants in time. At each instant, input data is acquired from the execution environment; output values are produced according to the system of equations considered as a network of operations.

Kernel of the SIGNAL *language.* It is based on four operations, defining primitive processes or equations, and a composition operation to build more elaborate processes in the form of systems of equations.

- **Functions** are transformations on data at an instant t. For example, the definition of a signal Y_t by the function f: $\forall t,\ Y_t = f(X_{1_t}, X_{2_t}, \ldots, X_{n_t})$ is encoded in SIGNAL: `Y := f{ X1, X2,..., Xn}`. The signals `Y`, `X1`,..., `Xn` are constrained to have the same clock.
- **Selection** of a signal `X` according to a boolean condition `C` is: `Y := X when C`. If `C` is present and *true*, then `Y` has the presence and value of `X`. The clock of `Y` is the *intersection* of (*i.e., included* in) that of `X` and that of `C` at the value *true*.
- **Deterministic merge** noted: `Z:= X default Y` has the value of `X` when it is present, or otherwise that of `Y` if it is present and `X` is not. Its clock is the *union* of (*i.e., includes*) or *contains* those of `X` and `Y`.
- **Delay** gives access to past values of a signal. *e.g.*, the equation $ZX_t = X_{t-1}$, with initial value V_0 defines a *dynamical process*. It is encoded by: `ZX := X$1` with initialization `ZX init V0`. `X` and `ZX` have equal clocks.

– **Composition** of processes is noted "|" (for processes P_1 and P_2, with parentheses: `(| P1 | P2 |)`). It consists in the composition of the systems of equations; it is associative and commutative. It can be interpreted as parallelism between processes; communication between them is carried by the broadcasting of signals.

Derived processes have been defined on the base of the primitive operators, providing programming comfort and modularity, *e.g.*, the instruction `synchro{X,Y}` specifies that signals `X` and `Y` are synchronous (*i.e.*, have equal clocks); `when B` gives the clock of *true*-valued occurrences of logical signal `B`; `X cell B` memorizes values of `X` and also outputs them when `B` is true. Arrays of signals and of processes have been introduced as well. Hierarchy and re-use of the definition of processes are supported by the possibility of defining process models that can be invoked by instantiation.

Time intervals and preemptive tasks. A recent extension to SIGNAL (SIGNAL*GTi*) handles tasks executing on time intervals and their sequencing and preemption [13]. The notion of *time interval* has been introduced: it is entered (takes the value `inside`) upon the occurrence of the start event, and is exited (takes the value `outside`) upon the occurrence of an end event, and can then be entered again, iteratively. Intervals have an initial state given by declaration. An interval is constructed by the statement: `I:= ]B,E] init I0` with initial value `I0` (`inside` or `outside`). With this extension, we can define the notion of *task* on an interval, which is a SIGNAL process active when the interval is `inside`, and inactive `outside`. A *suspensive task* is written `P on I`: it re-starts at its current state when re-entering `I`. An *interruptible task* is written `P each I`: it re-starts at its *initial state* (as defined by the declarations of its state variables). Processes can themselves be decomposed into sub-tasks: this way, the specification of *hierarchies* of preemptive behaviors is possible.

This extension is implemented as a pre-processor to the SIGNAL compiler, and is fully compatible with the environment, including the verification tools. In particular, the intervals are coded by a boolean state variable, `true` when the interval is `inside` and `false` when `outside`. Occurrences of a signal `X` inside an interval `I` are coded by `X when I`. The specification of the power transformer station uses this extension [11]. This kind of specification, using tasks and intervals, is useful to specify properties such as "two process are not active at the same time". An example is given in Section 4.2.

Verification tools for SIGNAL *programs.* The verification of a SIGNAL program can concern invariant properties (to be satisfied at all instants of its execution) or dynamical properties (to be satisfied on the histories of the program). The first kind is addressed by the compiler, which checks the consistency of constraints between the clocks and proves these statical properties. Different phases occur during the compilation of a SIGNAL program. One of these consists in the resolution of a system of boolean equations, coding the constraints among the different clocks. This clock calculus relies on an algebra on sets of instants detailed in [2]. In fact, the compiler has to check the consistency on the constraints on the clocks of the different signals of a SIGNAL program. This way, by composing the specification with the expression in SIGNAL of a statical property (*i.e.*, *temporally invariant* property), the compiler

checks if they are consistent w.r.t. each other. If so, their composition constitutes a correct controller that satisfies the property. An example is given in Section 4.2.

The second kind of properties is addressed by a formal method, based on a model of the behavior of the program presented in Section 2.2, and with which dynamical properties of the system can be proved.

2.2 An equational model of the behaviors of SIGNAL programs

The equational nature of the SIGNAL language leads naturally to the use of a method based on systems of polynomial dynamical equations over $\mathbb{Z}/_{3\mathbb{Z}}$ as a formal model of programs behavior. The systems of polynomial equations characterize sets of solutions, which are states and events. The method consists in manipulating the equation systems instead of the solutions sets, avoiding the enumeration of the state space. This paper makes an overview presentation of results without recalling details and proofs (see [9, 5, 8]).

Signals. In order to model its behaviors, a SIGNAL process is translated into a system of polynomial equations over $\mathbb{Z}/_{3\mathbb{Z}}$, *i.e.* integers modulo 3: {-1,0,1} [8]. The principle is to code the three possible states of a boolean signal `X` (*i.e.*, *present* and *true*, or *present* and *false*, or *absent*) in a *signal variable* x by:

$$\begin{cases} present \wedge true & \rightarrow +1 \\ present \wedge false & \rightarrow -1 \\ absent & \rightarrow 0 \end{cases}$$

For the non-boolean signals, we only code the fact that the signal is *present* or *absent*: $\begin{cases} present \rightarrow \pm 1 \\ absent \rightarrow 0 \end{cases}$. Note that the square of a *present* signal is 1, whatever its value. Hence, for a signal `X`, its clock can be coded by x^2. Thus, two synchronous signals `X` and `Y` satisfy the constraint equation: $x^2 = y^2$. This fact will be used extensively in the following.

Primitive processes. Each of the primitive processes of SIGNAL can be encoded in a polynomial equation. For example `C := A when B`, which means "*if* $b = 1$ *then* $c = a$ *else* $c = 0$" can be rewritten in $c = a(-b - b^2)$: the solutions of this equation are the set of possible behaviors of the primitive process **`when`**.

The delay `$`, which is a dynamical operator, is different because it requires memorizing the past value of the signal into a *state variable* ξ. In order to encode `Y := X$1 init Y0`, we have to introduce the three following equations:

$$\begin{cases} \xi' = x + (1 - x^2)\xi & (1) \\ y = \xi x^2 & (2) \\ \xi_0 = y_0 & (3) \end{cases}$$

Equation (1) describes what will be the next value ξ' of the state variable. If x is *present*, ξ' is equal to x (because $(1-x^2) = 0$), otherwise ξ' is equal to the last value of x, memorized by ξ. Equation (2) gives to y the last value of x (*i.e.* the value of ξ) and constrains the clocks y and x to be equal. Indeed, $y^2 = \xi^2 x^4$, and in $\mathbb{Z}/_{3\mathbb{Z}}$ we have $x^3 = x$, *i.e.*$x^4 = x^2$, so this leads to $y^2 = \xi^2 x^2$; as $\xi^2 = 1$ (because ξ is always

present), we finally get $y^2 = x^2$. Equation (3) corresponds to the initial value of ξ, which is the initial value of y.

Table 1 shows how all the primitive operators are translated into polynomial equations.

Boolean instructions	
`Y := not X`	$y = -x$
`Z := X and Y`	$z = xy(xy - x - y - 1)$ $x^2 = y^2$
`Z := X or Y`	$z = xy(1 - x - y - xy)$ $x^2 = y^2$
`Z := X default Y`	$z = x + (1 - x^2)y$
`Z := X when Y`	$z = x(-y - y^2)$
`Y := X $1 (init` y_0`)`	$\xi' = x + (1 - x^2)\xi$ $y = x^2\xi$ $\xi_0 = y_0$
non-boolean instructions	
`Y :=` $f(X_1, \ldots, X_n)$	$y^2 = x_1^2 = \cdots = x_n^2$
`Z := X default Y`	$z^2 = x^2 + y^2 - x^2y^2$
`Z := X when Y`	$z^2 = x^2(-y - y^2)$
`Y := X $1 (init` y_0`)`	$y^2 = x^2$

Table 1. Translation of the primitive operators.

Processes. By composing the equations representing the elementary processes, any SIGNAL specification can be translated into a set of equations called polynomial dynamical system. Using this encoding, the reaction events of the program, *i.e.* the value of each of the m *signal variables* and n *state variables*, are represented by a vector in $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$. Formally, a polynomial dynamical system can be reorganized into three sub-systems of polynomial equations of the form:

$$\begin{cases} Q(X,Y) = 0 \\ X' = P(X,Y) \\ Q_0(X) = 0 \end{cases}$$

where:

- X is a set of n variables, called *state variables*, represented by a vector in $(\mathbb{Z}/_{3\mathbb{Z}})^n$;
- Y is a set of m variables, called *event variables*, represented by a vector in $(\mathbb{Z}/_{3\mathbb{Z}})^m$;
- $X' = P(X,Y)$ is the *evolution equation* of the system; it can be considered as a vectorial function $[P_1, \ldots, P_n]$ from $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$ to $(\mathbb{Z}/_{3\mathbb{Z}})^n$. It groups all the equations on the state variables, and characterizes the dynamical aspect of the system ;

- $Q(X,Y) = 0$ is the *constraints equation* of the system, it is a vectorial equation $[Q_1, \ldots, Q_l]$. It groups all the equations characterizing the statical aspect of the system (invariant for all instants t) ;
- $Q_0(X) = 0$ is the *initialization equation* of the system, it is a vectorial equation $[Q_{0_1}, \ldots, Q_{0_n}]$. It groups all the equations characterizing the initialization of the system.

For example the following small process in SIGNAL,

```
process altern =
{? event A,B
 !}
(| X := not ZX
 | ZX := X$1
 | synchro{A,when X}
 | synchro{B,when ZX}
 |)
where
 logical X, ZX init false
end
```

is translated in the polynomial dynamical system with variable a, b, X and zx corresponding to the events A, B and the logical signals X and ZX and a state variable *state* introduced by the delay. The system consist of

- an initialization equation : $state = -1$,
- an evolution equation : $state' = x + (1 - x^2) * state$
- and a system of constraint equation

$$x = -zx, zx = state * x^2, a^2 = when\ x, b^2 = when\ zx$$

A polynomial dynamical system can be seen as a finite transition system. The initial states of this automaton are the solutions of the equation $Q_0(X) = 0$. When the system is in a state $x \in (\mathbb{Z}/_{3\mathbb{Z}})^n$, any event $y \in (\mathbb{Z}/_{3\mathbb{Z}})^m$ such that $Q(x,y) = 0$ can fire a transition. In this case, the system evolves to a state x' such that $x' = P(x,y)$.

Using this kind of representation and the operations explained in the next section, we hope for the reduction of combinatoric explosion experienced with automata composition. The number of states of the resulting automata is the product of the number of states of each automaton. In contrast, the composition of two polynomial dynamical systems is simply obtained by putting the equation together.

We thus have a mathematical model characterizing the behavior of dynamical systems. In the perspective of analyzing these behaviors by the evaluation of the satisfaction of properties, we need operations on polynomial systems, which correspond to the manipulation of the sets of their solutions. This way we can express ourselves about sets of behaviors, states and transitions, while still remaining in the domain of polynomial functions, and not having to enumerate them.

3 Verifying SIGNAL programs

3.1 Operations on the polynomial dynamical systems

The theory of polynomial dynamical systems uses operations in algebraic geometry such as varieties, ideals and morphisms. They are used to define the properties of systems such as liveness, invariance and invariance under control. This section presents the essential results of an extensive study [9, 8].

Description of the basic objects and operations. Let us define the quotient ring of polynomial functions $A[X,Y] = \mathbb{Z}/_{3\mathbb{Z}}[X,Y]/_{(X^3-X,Y^3-Y)}$ [2]: it is the set of polynomials in $\mathbb{Z}/_{3\mathbb{Z}}$ for which the degree in each variable is ≤ 2 because of the fact that $X^3 = X$. Let E be a set of event and state variables in $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$. The following set of polynomials:

$$I(E) = \{p \in A[X,Y] \ / \ \forall (x,y) \in E,\ p(x,y) = 0\}$$

is called the *ideal* of E in $A[X,Y]$. This set represents all the polynomials, for which the set E is a solution. In terms of dynamical systems, it represents the set of equations characterizing the states and events in E.

Reciprocally, to any set of polynomials G, we can associate a set in $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$, called the *variety* of G, defined as follows:

$$V(G) = \{(x,y) \in (\mathbb{Z}/_{3\mathbb{Z}})^{n+m} \ / \ \forall p \in G,\ p(x,y) = 0\}$$

This set represents all the solutions for a given set of polynomials. In terms of dynamical systems, it represents the set of states and events admissible by the dynamical systems in G.

The advantage of using ideals is that there exists a direct correspondence between an ideal and the associated variety. In fact, we can easily prove that, in the quotient ring $A[X,Y]$:

$$V(I(E)) = E \text{ and } I(V(< G >)) =< G >$$

where, for a set of polynomials G, $< G >$ is the set of all linear combinations of polynomials in G: this means that their solutions include those of G. This way, we can translate properties of sets into equivalent properties of associated ideals of polynomials. Hence, instead of manipulating explicitly and enumerating the states, this approach manipulates the polynomial functions characterizing their sets. An other important aspect is that an ideal can be represented by a single polynomial, called *the principal generator*. This particularity is used in the practical implementation of the algorithms on ideals [5].

For example, for the constraint equation Q of a polynomial dynamical system: the equation $Q(X,Y) = 0$ represents a set of polynomial equations, decomposed as follows:

$$\begin{cases} Q_1(X,Y) = 0 \\ \cdots \\ Q_l(X,Y) = 0 \end{cases}$$

[2] $X^3 - X$ (resp. $Y^3 - Y$) denotes all the polynomials $X_i^3 - X_i$ (resp. $Y_i^3 - Y_i$).

If E is the set of solutions of this system of equations, it is clear that

$$E = V(< Q_1, ..., Q_l >) \text{ and } I(E) =< Q_1, ..., Q_l > .$$

So instead of manipulating the set of solutions E of the constraint equation, represented in our case by a variety, we can easily convert it into an ideal $I(E)$, which can be represented by a single polynomial. Thus, the relations between different sets (*e.g.* inclusion or projection), can be translated into operations on polynomials.

Operations on dynamical behaviors. To capture the dynamical aspect of a polynomial dynamical system, we introduce the notion of morphism and comorphism. A *morphism* (often called in other community *post-condition*) is a polynomial function P from $(\mathbb{Z}/_{3\mathbb{Z}})^{n+m}$ to $(\mathbb{Z}/_{3\mathbb{Z}})^n$ (the *evolution equation* $X' = P(X, Y)$ of the system, for example).

With the morphism P, there is an associated *comorphism* P^* from $\mathbb{Z}/_{3\mathbb{Z}}[X]$ to $\mathbb{Z}/_{3\mathbb{Z}}[X, Y]$, defined by:

for a polynomial $p \in \mathbb{Z}/_{3\mathbb{Z}}[X]$:

$$\begin{aligned} P^*(p(X)) &= P^*(p(X_1, ..., X_n)) \\ &= p(P_1(X, Y), ..., P_n(X, Y)) \end{aligned}$$

where $P_1, ..., P_n$ are the components of P. In other words $P^*(p(X))$ is obtained by substituting every X_i in p with the corresponding $P_i(X, Y)$. In fact, the comorphism (which is often called *pre-condition*) can be seen as a map computing the states *from which* we can reach the states that are solutions of the *evolution equations*; it can be used to take the transitions backwards.

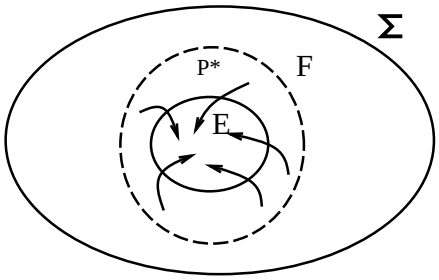


Fig. 1. Representation of the comorphism P^*

In Figure 1, F represents all the states from which all the states of E can be reached with only one transition (F represents the set obtained by the application of the comorphism on E), where Σ represents the set of states of the system.

This is the basic tool for analyzing transitions between states. But, we do not have to compute this transition map, which has a very high computing complexity. There are relations between varieties and ideals using morphisms and comorphisms that are used to perform calculations on the properties of polynomial dynamical systems.

3.2 Properties on the polynomial dynamical systems

Various properties of systems can be evaluated on such models using these operations; in this subsection we define several properties and give their expression in terms of algebraic operators, such as they are established in [5].

Liveness. We say that a system is *alive* if and only if it can not be in a state from which no transition can be taken, *i.e.* no deadlock can occur. This property states that every trajectory of the system is infinite. In terms of polynomial dynamical systems, this definition can be formalized as follows:

Definition 1.
- A *state* x is alive if there exists a signal y such that $Q(x, y) = 0$ (*i.e.*a transition can be taken) ;
- A *set of states* V is alive if and only if every state of V is alive ;
- A *system* is alive, if and only if $\forall(x, y)$ such that $Q(x, y) = 0$, $P(x, y)$ is an alive state (*i.e.*, from live states, only live states can be reached).

Using this definition, it can be proved [5] that the property of liveness of a system can be stated as follows: $P^*(< Q > \cap \mathbb{Z}/_{3\mathbb{Z}}[X]) \subseteq < Q >$

Safety. Informally, whereas a liveness property stipulates that some *good things* do happen, a safety property stipulates that some *bad things* do not happen during any execution of the program [1]. In our case this kind of property covers the class of properties, describing the set of good states which remains invariant. The definition of an invariant set of states is as follows:

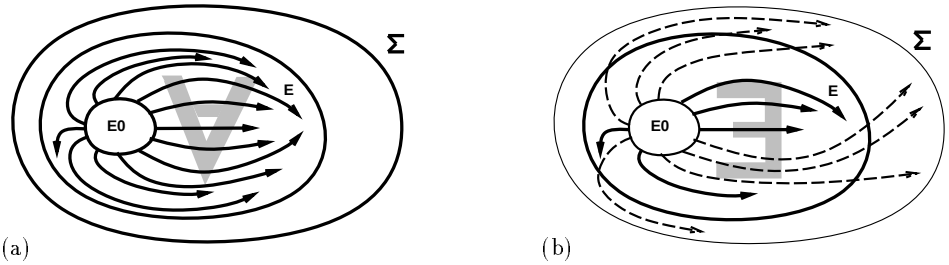
Definition 2. A subset E of states is *invariant* for a dynamical system, if and only if for every state $x \in E$ and for *every event* y admissible in the state x, the state $x' = P(x, y)$ is in E.

This way, if we describe a property by an equivalent set of states which verify it, the property is always verified if and only if this equivalent set of states is invariant for the dynamical system.

Using this definition, it has been proved [5] that the invariance of a property, represented by a set of states E, considering a polynomial dynamical system, can be stated as: $< P^*(I(E)) > \subseteq < Q > + I(E)\mathbb{Z}/_{3\mathbb{Z}}[X, Y]$. This notion is illustrated in Figure 2(a). Σ is the set of the states of the system; the set E_0 is the set of initial states, and E represents the set of states, which verify the property. The arrows correspond to the different possible trajectories of the system, which remain inside the set E, because it is invariant.

It may happen that the property, represented by an equivalent set of states is not invariant. In this case, it is interesting to compute the largest invariant subset included in the set of states E . This property is evaluated using a fix-point computation.

Definition 3. A subset E of states is control-invariant for a dynamical system, if and only if for every state $x \in E$, *there exists an event* y admissible in the state x, the state $x' = P(x, y)$ is in E.



(a) (b)

Fig. 2. Invariance (a) and invariance under control (b) of E (from E_0)

Using Definition 3, it has been proved [5] that a sub-set E is control-invariant for a dynamical system, if and only if: $(< Q > + P^*(I(E))) \cap \mathbb{Z}/_{3\mathbb{Z}}[X] \subseteq I(E)$. This notion is illustrated in Figure 2(b). The dotted arrows correspond to the trajectories of the system, which should be forbidden or inhibited by a controller in order to obtain invariance. It is also possible to compute the largest control invariant sub-set included in a given set E of states.

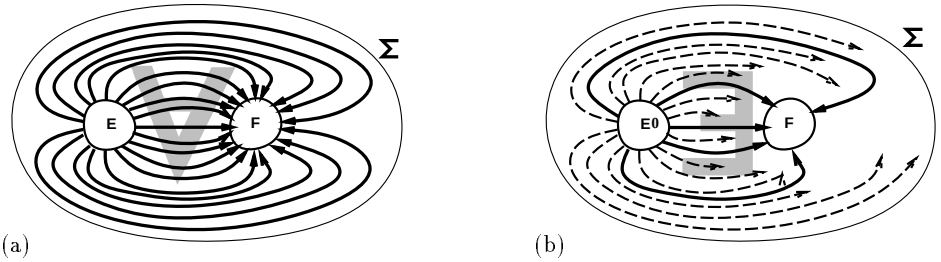
Other kinds of properties may be derived from the liveness, invariance and control invariance properties.

Reachability and attractivity.

Definition 4. A subset F of states is reachable for a dynamical system, if and only if every state $x \in F$ can be reached from the initial states E_0 of the considered dynamical system *i.e.*, there exists a trajectory initialized in E_0 that reaches x.

To prove this property, we use the largest invariant sub-set of a set, as described before. Thus, a set of states F is reachable from the initial states of a polynomial dynamical system if and only if the initial states are *not* included in the largest invariant sub-set of the *complement* of F (*i.e.*, from the initial states, one is not compelled to stay in states not verifying the property).

This notion is illustrated in Figure 3(b). The black arrows represent the trajectories of the system, which reach the set of states F, whereas the dotted arrows correspond to the trajectories, which never reach the set of states F.



(a) (b)

Fig. 3. Attractivity (a) and reachability (b) and of F w.r.t. E

Definition 5. A set of states F is attractive for a set of states E, if and only if, every trajectory initialized in E reaches F.

Using the definition above, we can prove that F is attractive for E if the set E is not included in the greatest control-invariant of the *complement* of F (*i.e.*, a trajectory can not lead to an invariant set, which does not contain the set F, and from which it is impossible to reach F). This notion is illustrated in Figure 3(a).

This section made an overview of the method for the verification of properties, with its basic operators: set theoretic operators, fix-point computation, quantifiers elimination. Using the algebraic methods, explained before, it is also possible to express CTL formulae. The reader interested in the theoretical foundation of this approach is referred to [9, 5, 8].

4 Application to a power transformer station

4.1 Specification of the power transformer station

The transformer stations on the power network. The French national power network, operated by *Électricité de France* (EDF), counts a large number of transformer stations. For each high voltage line, a transformer lowers the voltage, so that it can be distributed in urban centers to end-users [11]. In the course of exploitation of this system, several kinds of electrical defects can occur, due to causes internal or external to the station. Three types of electrical defects are considered: phase (`PH`), homopolar (`H`), or wattmetric (`W`). In order to protect the device and the environment, several circuit breakers are placed in different parts of the station. These circuit breakers are alerted by sensors at different locations, and controlled by local control systems called *cells* (arrival cell, link cells, and departure cells, see Figure 4) and by an operator in a remote control center.

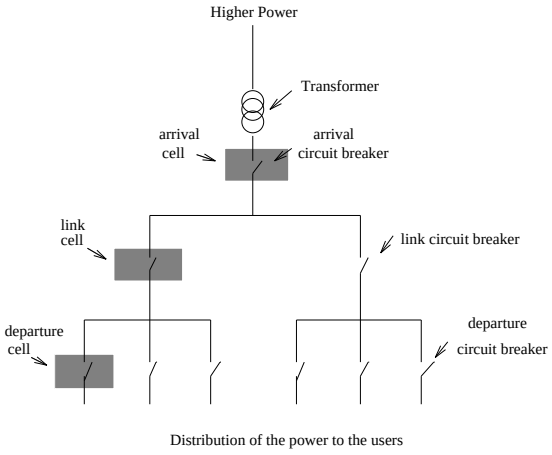


Fig. 4. Topology of a power transformer station

Functional description of a departure cell. We will focus on one of these types of cell: the departure cell, because it features all the interesting aspects of the controllers behavior. It is decomposed in a confirmation process, which sequentially tests for the different types of defect, followed by a treatment phase, consisting in attempts at making the defect disappear. These behaviors feature sub-tasks which are interrupted, in a nested way, and are repeated on series of activity intervals; their specification makes use of the corresponding constructs of SIGNAL*GTi* [11]. We only describe here the details necessary for the understanding of the verification presented further.

The *confirmation phase* consists in detecting the first defect occurrence (event `First_Defect`) and from then on, for each of the defect types (`PH`, `H`, or `W`), and in taking the time to let transient defects cease naturally, or else assess their persistent presence. They are tested in sequence: from `First_Defect`, interval `I_PH` is entered. Associated with this interval, the confirmation task first waits until `Delay_PH` is elapsed, and then enters interval `I_H` in which a task first waits until `Delay_H` is elapsed, and then enters interval `I_W` in which a task waits until `Delay_W` is elapsed. If in the meantime a defect is confirmed (*i.e.*, `PH`, `H` or `W` is present at the end of the corresponding delay), the sequence is interrupted (interval `I_PH` is ended), and the defect is confirmed by emission of the boolean `Def_Conf` with value `true`. It is also exited if the defect disappears: the last delay elapses without defect (with `Def_Conf` at value `false`)emission of event `Ext_Def`)A first property to be verified is that if a defect is detected after the end of its corresponding delay then the defect is actually confirmed (*i.e.*, in the example of a `PH`-defect, if `PH` is present in the interval `I_H`, then `Def_Conf` is actually emitted). A second property is that the confirmation is never active at the same time as the treatment (*i.e.* the controller can not be in states where both intervals `I_PH` and `I_Treat` are `inside`). A third property is that if a defect appears then the controller will necessarily evolve in such a way that either the defect is confirmed, or it disappears, or an external defect occurs (*i.e.* the controller necessarily evolves towards states where one of these is true).

The *treatment phase* begins when the defect is confirmed: the interval `I_Treat` is entered on the occurrence of `Def_Conf`. The task alternates breaking the circuit during varying delays, and closing it again to check whether the defect has disappeared, for a certain number of cycles. At the end of the last cycle, if the defect is present the circuit breaker is definitely cut off: the signal `Def_Break` is emitted, and the management is left to a remote control operator. A property to be verified here is that if a defect is confirmed, then either it disappears and the circuit-breaker is closed, or it does not and `Def_Break` is emitted.

4.2 Formal verification of the power transformer station

In this section, we apply the differents tools we have presented to check various properties on our SIGNAL implementation of the power transformer station. After the translation of our SIGNAL program, we obtain a dynamical system (This translation has been made in 10s. During this same time, we have also checked the causal and temporal concurrency of our program and produced an executable code). The polynomial dynamical system obtained is represented by 12 state variables and 22 event variables; that is to say, our system can be represented by an automaton of

500.000 possibles states. In fact, we must just consider the number of reachable states. For that, we just have to compute the orbit of the system, corresponding to the set of all the states which can be reached from the initial states. Using our representation in ideals and varieties, this set is represented by a single polynomial. Then to obtain the number of different states, we have to count the number of solutions of our polynomial.In our case, the system can involve in 7000 different reachable states.

We will now describe the different properties, which have been proved.

1. *if a defect* `PH` *is detected after the end of its corresponding delay, in interval* `I_H`*, then the defect is confirmed by the emission of* `Def_Conf`*.*
   To check this property, we use the SIGNAL compiler, which proves statical properties (*i.e.*, invariant over time instants) as explained in Section 2. We express the property in SIGNAL as an inclusion between clocks as follows:

   ```
   synchro{(when PH when I_H), ((when PH when I_H) when Def_Conf)}
   ```

   (where $A \subset B$ is expressed in the form $A = A \cap B$, with `when` as intersection, and `synchro` as equality for clocks). We compose this constraint with the controller. The compilation of the whole checks the consistency of all the constraints on clocks in the specification. The fact that our composed specification is consistent means that this new constrained controller verifies the property.
2. *the controller can not be in states where both intervals* `I_PH` *and* `I_Treat` *are* `inside`*.*
   This property can be established by proving that the set of states corresponding to the situation where the treatment phase and the confirmation phase are both active, can not be reached from the initial states of the controller (given by the declarations in the program).
   For that, we consider the two intervals `I_Treat` and `I_PH`, encoded by logicals which are `true` when the system is in the treatment phase, respectively in the confirmation phase. After the translation of the SIGNAL program into the corresponding polynomial dynamical system over $\mathbb{Z}/_{3\mathbb{Z}}$, we compute the set of states where `I_Treat`=1 and `I_PH`=1. Then, the method consists in verifying that this set of states is not reachable from the initial states of the polynomial dynamical system. Reachability can be computed as described in Section 3.2 by a function of the proof system: in our example the result obtained is *false*.
3. *If* `First_Defect` *occurs,then the controller will necessarily evolve in such a way that:*
   (a) either `Def_Conf` *will be emitted with value* `true`
   (b) or `Def_Conf` *will be emitted with value* `false`*.*
   (c) or event `Ext_Def` *occurs.*
   The method for verifying this property is to build an observer, which is a process composed with the controller, and which evaluates a boolean signal `OUT`, which is *present* when one of these possibilities occurs, `true` when the conditions (a) or (c) are verified, and `false`, when the condition (b) is verified. The property can be proved by checking the attractivity of the set of states F, where `OUT` is present, from the set of states E, where the defect appears (*i.e.* there is an occurrence of the event `First_Defect`: `First_Defect`=1). By applying the proof

system function computing attractivity as described by Definition 5, we prove that F is attractive from E (*i.e.* when a defect appears, all the trajectories of the system lead to the state where `OUT` is present).

4. *If* `Def_Conf` *occurs, then the controller will necessarily evolve in such a way that:*

   (a) either the defect does not disappear and the signal `Def_Break` *will be emitted,*
   (b) or the defect does disappear, with the circuit-breaker closed.

   The method used to prove this property is the same as that used to prove the property (3). We compute the set of states E, where the defect is confirmed (*i.e.* `Def_Conf`=1), and the set of states F, corresponding to the union of the states where the condition (a) is verified and the states where the condition (b) is verified. Using the function computing attractivity, we prove that F is an attractive set of states from the set of states E.

5 Conclusion

This paper presents the verification method associated with the SIGNAL reactive language, and its application to the controller of a power transformer station, that was specified and implemented in SIGNAL and its extention with nested preemptive tasks SIGNAL*GTi* [11]. The verification is based on the model underlying SIGNAL, *i.e.* systems of polynomial dynamical equations over $\mathbb{Z}/_{3\mathbb{Z}}$ [9]. The systems of polynomial equations characterize a set of solutions which encode states and events. The techniques used in the method consist in manipulating the equation systems instead of the solutions sets, which can avoid the enumeration of the state space. Operations used on equations systems belong to the theory of algebraic geometry, such as varieties, ideals and morphisms. They enable the treatment of properties of *safety*, *liveness*, *reachability* and *attractivity*. The SIGNAL approach to the verification of control systems has also been experimented on other applications, such as a robotic production cell [3].

The equational nature of the SIGNAL language makes it natural to use an equational framework for modeling behaviors and proving properties on them. This description of dynamical systems using equations is quite common in the fields of control theory and digital circuits, but not in verification and model-checking. This aspect is an originality of the SIGNAL approach compared to others using transition systems. For example, the reactive languages ESTEREL [4] and LUSTRE [7] are compiled into finite state automata; hence they naturally interface with tools based on these formalisms like AUTO and AUTOGRAPH. In principle the two methods are equivalent, but in practice they might be better suited each to a certain class of problems; in particular, the compactness of the implicit representation by a system of equations can help avoiding the combinatorial explosion of explicit state-based representations. Both models support verification by the methods of model-checking and comparison (bisimulation or behavioral equivalence), and as in the case of LUSTRE, some properties or observers can be specified in the language. Given that polynomial dynamical systems are an implicit description of transition systems, it is possible to give a semantics of temporal logic formulae (for example the Computational Tree Logic CTL) in terms of the algebraic operators [5], and perform symbolic model

checking by evaluating them on a polynomial model. We can notice that the language LUSTRE uses the same methodology for the verification of their programs, using Binary Decisions Diagrams to encode their formulas [7].

A perspective for a different use of the polynomial model is the automated synthesis of controllers, where algebraic methods are used for the derivation, from a model of a system, of a controller statisfying given properties and objectives such as invariance or attractivity [6, 12]. In our application, the method will be used for the synthesis of the controller of the interactions between the various cells composing the transformer station controller. Another perspective concerns the possibility of proving properties that depend on the behavior of numerical variables, or in general on data other than presence/absence and Boolean which are handled currently.

References

1. B. Alpern and F. B. Schneider. – Recognizing safety and liveness. – Technical Report 86-727, Departement of Computer Science Cornell University, Ithaca, New York, January 1986.
2. T. A. Amabegnon, L. Besnard, and P. Le Guernic. – Arborescent canonical form of booelan expressions. – Technical Report 2290, INRIA, June 1994. – (ftp: **ftp.inria.fr**, file **/INRIA/publications/RR/RR-2290.ps.Z**).
3. T. Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten. – SIGNAL – *the specification of a generic, verified production cell controller*, volume 891 of *LNCS (Lecture Notes in Computer Science)*, chapter VII, pages 115 – 129. – Springer Verlag, January 1995.
4. F. Boussinot and R. de Simone. – The ESTEREL language. – *Proc. of the IEEE*, 9(79):1293–1304, September 1991.
5. B. Dutertre. – *Spécification et preuve de systèmes dynamiques: Application à* SIGNAL. – Thèse, Université de Rennes, December 1992. – (In French).
6. B. Dutertre and M. Le Borgne. – Control of polynomial dynamic systems: an example. – Technical Report 2193, INRIA, January 1994. – ftp: **ftp.inria.fr**, file **/INRIA/publications/RR/RR-2193.ps.Z**.
7. N. Halbwachs, F Lagnier, and C. Ratel. – Programming and verifying real-time systems by means of the synchronous data-flow programming language LUSTRE. – In *IEEE Transactions on Software Engineering, Special issue on the Specification and Analysis of Real-Time Systems*, September 1992.
8. M. Le Borgne. – *Systèmes dynamiques sur des corps finis.* – Thèse, Université de Rennes I, September 1993. – (In French).
9. M. Le Borgne, A. Benveniste, and P. Le Guernic. – Dynamical systems over galois fields and deds control problems. – In *Proc. of 33^{t}h IEEE Conf. on Decision and Control*, volume 3, pages 1505–1509, 1991.
10. P. Le Guernic, M. Le Borgne, T. Gautier, and C. Le Maire. – Programming real time application with SIGNAL. – *Proc. of the IEEE*, 79(9):1321–1336, September 1991.
11. H. Marchand, E. Rutten, and Samaan M. – Specifying and verifying a transformer station in SIGNAL and SIGNAL*GTi*. – Technical Report 2521, inria, March 1995. – (ftp: **ftp.inria.fr**, file **/INRIA/publications/RR/RR-2521.ps.Z**).
12. P.J. Ramadge and W.M. Wonham. – The control of discrete events systems. – *Proc. of the IEEE*, 77(1):81–97, January 1989.
13. E. Rutten and P. Le Guernic. – The sequencing of data flow tasks in SIGNAL. – In *Proceedings of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, Orlando, Florida, June 1994. –

Table of Contents

1 Introduction 1

2 The SIGNAL language and its model 2
2.1 The SIGNAL equational language 2
Kernel of the SIGNAL language. 2
Time intervals and preemptive tasks. 3
Verification tools for SIGNAL programs. 3
2.2 An equational model of the behaviors of SIGNAL programs 4
Signals. 4
Primitive processes. 4
Processes. 5

3 Verifying SIGNAL programs 7
3.1 Operations on the polynomial dynamical systems 7
Description of the basic objects and operations. 7
Operations on dynamical behaviors. 8
3.2 Properties on the polynomial dynamical systems 9
Liveness. 9
Safety. 9
Reachability and attractivity. 10

4 Application to a power transformer station 11
4.1 Specification of the power transformer station 11
The transformer stations on the power network. 11
Functional description of a departure cell. 12
4.2 Formal verification of the power transformer station 12

5 Conclusion 14

List of Figures

1 Representation of the comorphism P^* 8
2 Invariance (a) and invariance under control (b) of E (from E_0) 10
3 Attractivity (a) and reachability (b) and of F w.r.t. E 10
4 Topology of a power transformer station 11

List of Tables

1 Translation of the primitive operators. 5

This article was processed using the LaTeX macro package with LLNCS style